Mini-Chapter — Reproducible Tooling for Applied Financial Engineering

Financial engineering is sensitive to small changes in code, data, and environment. Reproducibility is the discipline that prevents divergence between your results and teammates', regulators', or future you.

1. The Reproducibility Triangle

- **Code**: notebooks and modules (`src/`)
- **Data**: inputs/outputs (`/data/`)
- **Environment**: Python version + packages

To reproduce a result, all three must be controlled. Environments isolate versions; Git tracks code and instructions; data locations are parameterized via configuration.

2. Isolated Python Environments

Use **conda** or **venv** to avoid dependency collisions.

- Conda: `conda create -n fe-course python=3.11 -y && conda activate fe-course`
- venv: `python -m venv env && source env/bin/activate`

Pin packages with `pip freeze > requirements.txt`. Record Python version in README. Advanced: export conda `environment.yml` using `--from-history`.

3. Secrets & Configuration with `.env`

Hardcoding secrets/paths in notebooks is risky and brittle. Instead:

- Create `.env.example` committed to the repo (no secrets, only placeholders).
- Keep a real `.env` locally (ignored by Git).
- Load values with `python-dotenv` and access via `os.getenv()`.

Example:

```
# .env.example
API_KEY=your_dummy_key_here
DATA_DIR=./data
```

```python
from dotenv import load_dotenv
import os
load_dotenv()
api_key = os.getenv("API_KEY")
```

4. Project Structure

A simple, maintainable scaffold:

```
project/
├─ data/          # raw/processed data (often gitignored)
├─ notebooks/     # exploratory & lecture notebooks
├─ src/           # reusable modules (e.g., config.py)
├─ reports/       # optional outputs
├─ requirements.txt
├─ .env.example   # template; .env stays local
```

5. Jupyter as a Literate Environment

- Mix **markdown** with code to document intent.
- Use **Restart & Run All** to ensure linear, deterministic execution.
- Prefer `src/` modules for reusable logic; notebooks for narrative.

6. Version Control with Git/GitHub

- Initialize early and commit often with clear messages.
- Add `.gitignore` entries for `.env`, caches, and large data.
- Use branches/PRs for features and reviews.

7. Troubleshooting & Pitfalls

- Wrong kernel/interpreter → reselect kernel to your env.
- Mixed package managers → stick to one flow per environment.
- Accidental secret commit → immediately rotate the key; add to `.gitignore`.

8. How This Stage Feeds the Course

Today’s outputs (scaffold, `.env`, requirements, first notebook) become the backbone for future EDA, feature engineering, modeling, and deployment.